Invoking Java Code with JSP Scripting Elements
Agenda

- Static vs. dynamic text
- Dynamic code and good JSP design
- JSP expressions
- Servlets vs. JSP pages for similar tasks
- JSP scriptlets
- JSP declarations
- Predefined variables
- Comparison of expressions, scriptlets, and declarations
Uses of JSP Constructs

Simple Application

- Scripting elements calling servlet code directly
- Scripting elements calling servlet code indirectly (by means of utility classes)
- Beans
- Servlet/JSP combo (MVC)
- MVC with JSP expression language
- Custom tags

Complex Application
Design Strategy: Limit Java Code in JSP Pages

• You have two options
  – Put 25 lines of Java code directly in the JSP page
  – Put those 25 lines in a separate Java class and put 1 line in the JSP page that invokes it

• Why is the second option *much* better?
  – Development. You write the separate class in a Java environment (editor or IDE), not an HTML environment
  – Debugging. If you have syntax errors, you see them immediately at compile time. Simple print statements can be seen.
  – Testing. You can write a test routine with a loop that does 10,000 tests and reapply it after each change.
  – Reuse. You can use the same class from multiple pages.
Which is better?

• JSP works best when the structure of the HTML page is fixed but the values at various places need to be computed dynamically

• If the structure of the page is dynamic, servlets may be more beneficial

• If the page has little static content, servlets are clearly superior
Basic Syntax

• HTML Text
  – `<H1>Blah</H1>`
  – Passed through to client. Really turned into servlet code that looks like
    • `out.print("<H1>Blah</H1>")`;

• HTML Comments
  – `<!-- Comment -->`
  – Same as other HTML: passed through to client

• JSP Comments
  – `<%-- Comment --%>`
  – Appears in the JSP page but not sent to client

• To get `<%` in output, use `<\%`
Types of Scripting Elements

• **Expressions**
  – Format: `<%= expression %>`
    • `<%= new java.util.Date() %>`
  – Evaluated and inserted into the servlet’s output.
    i.e., results in something like `out.print(expression)`

• **Scriptlets**
  – Format: `<% code %>`
  – Inserted verbatim into the servlet’s `_jspService` method
    (called by service)

• **Declarations**
  – Format: `<%! code %>`
  – Inserted verbatim into the body of the servlet class,
    outside of any existing methods
JSP Expressions

• Format
  – `<%= Java Expression %>`

• Result
  – Expression evaluated, converted to String, and placed into HTML page at the place it occurred in JSP page
  – That is, expression placed in _jspService inside out.print

• Examples (code in a .jsp file)
  Current time: `<%= new java.util.Date() %>`
  Your hostname: `<%= request.getRemoteHost() %>`
The autogenerated servlets use predefined variable names
- There are eight automatically defined local variables in jspService; known as implicit objects. The five below are the most commonly used.
- Not accessible in declarations or in utility classes invoked by JSP pages

**request**
- The HttpServletRequest (1st argument to service/doGet)

**response**
- The HttpServletResponse (2nd arg to service/doGet)

**out**
- The Writer (a buffered version of type JspWriter) used to send output to the client

**session**
- The HttpSession associated with the request (unless disabled with the session attribute of the page directive). Sessions are created automatically in JSP.

**application**
- The ServletContext obtained by getServletContext. Servlets and JSP pages can store persistent data in this object rather than in instance variables making it available to all servlets and JSP pages in the Web application.
JSP Expressions: Example

...<BODY>
<H2>JSP Expressions</H2>
<UL>
  <LI>Current time: <%= new java.util.Date() %>
  <LI>Server: <%= application.getServerInfo() %>
  <LI>Session ID: <%= session.getId() %>
  <LI>The <CODE>testParam</CODE> form parameter:
      <%= request.getParameter("testParam") %>
</UL>
</BODY></HTML>
• JSP expressions become print statements in the servlet that results from the JSP page
  – Instead of being placed in the doGet method, they are placed in a new method called _jspService that is called by service for GET and POST requests
  – Instantiates a JSPWriter
JSP/Servlet Correspondence

• Original JSP

```html
<H1>A Random Number</H1>
<%= Math.random() %>
<% Math.random() %>
```

• Representative resulting servlet code

```java
public void _jspService (HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    HttpSession session = request.getSession();
    JspWriter out = response.getWriter();
    out.println("<H1>A Random Number</H1>"Articles and Newsletters
    out.println(Math.random());
    Math.random();
    ...
}
```
public class ThreeParams extends HttpServlet {
    public void doGet
        (HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
            ...
            out.println(docType +
                    "<HTML>
                    <HEAD><TITLE>" + title + "</TITLE>
                    </HEAD>
                    <BODY BGCOLOR="#FDF5E6">
                    <H1 ALIGN="CENTER">" + title + "</H1>
                    <UL>
                        <LI><B>param1</B>: " + request.getParameter("param1") + "
                        <LI><B>param2</B>: " + request.getParameter("param2") + "
                        <LI><B>param3</B>: " + request.getParameter("param3") + "
                    </UL>
                    </BODY></HTML>"};
}
Reading Three Request Parameters

- param1: ~hall
- param2: ~gates
- param3: ~mcnealy
Reading Three Params (Servlet): Result

Reading Three Request Parameters

- param1: foo
- param2: bar
- param3: baz
JSP Scriptlets: Example

• Suppose you want to let end users customize the background color of a page

  `<BODY BGCOLOR="<%= request.getParameter("bgColor") %>"`

  – The syntax is correct but there is no validation of the input (missing or incorrect)
  – Scriplets can provide the solution
Scirplets

- **Scirplets let you insert arbitrary code into the servlet’s _jspService method**
  - They allow you to set response headers and status codes, execute code that has loops, conditionals
  - Takes the form `<% Java Code %>`
- **Have access to the predefined variables**
  - Explicitly send output to a page via a scirplet
    `<% String queryData = request.getQueryString();
    out.println("Attached GET data: " + queryData);
    %>`
  - Could also be done with scirplet and expression
    `<%String queryData = request.getQueryString(); %>`
    Attached GET data : `<%= queryData %>`
  - Or could be accomplished with only an expression
    `Attached GET data: `<%= request.getQueryString() %>``
<!DOCTYPE ...>
<html>
<head>
  <title>Color Testing</title>
</head>
<body bgcolor=""><%= bgColor %"></body>
</html>
JSP Scriptlets: Result

Testing a Background of "WHITE"

Testing a Background of "C0C0C0"

Testing a Background of "papayawhip"
Using Scriptlets to Make Parts of the JSP File Conditional

• **Point**
  – Scriplets are inserted into servlet *exactly* as written
  – Need not be complete Java expressions
  – Complete expressions are usually clearer and easier to maintain, however

• **Example**
  – `<%= if (Math.random() < 0.5) { %>
    Have a <B>nice</B> day!
  <% } else { %>
    Have a <B>lousy</B> day!
  <% } %>`

• **Representative result**
  – `if (Math.random() < 0.5) {
      out.println("Have a <B>nice</B> day!");
    } else {
      out.println("Have a <B>lousy</B> day!");
    }`
JSP Declarations

• You can define methods or fields that get inserted into the main body of the servlet class (outside the _jspService method)

• They don’t generate output but are normally used in conjunction with JSP expressions or scriplets

• Do not override the standard servlet life-cycle methods (service, doGet, init, etc.)
  – You can override jspInit and jspDestroy but not init or destroy

• In general, define most methods in Java classes not declarations (easier to test, debug and reuse)
JSP Declarations

• Format
  – `<%! Java Code %>`

• Result
  – Code is inserted verbatim into servlet's class definition, outside of any existing methods

• Examples
  – `<%! private int someField = 5; %>`
  – `<%! private void someMethod(...) {...} %>`

• Design consideration
  – Fields are clearly useful. For methods, it is usually better to define the method in a separate Java class.
JSP/Servlet Correspondence

• Original JSP

<H1>Some Heading</H1>
<%! 
    private String randomHeading() {
        return("<H2>" + Math.random() + 
            "</H2>");
    }
%>
<% = randomHeading() %>

• Possible resulting servlet code

```java
public class xxxx implements HttpJspPage {

    private String randomHeading() {
        return("<H2>" + Math.random() + "</H2>");
    }

    public void _jspService(HttpServletRequest request,
                            HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        HttpSession session = request.getSession();
        JspWriter out = response.getWriter();
        out.println("<H1>Some Heading</H1>"),
        out.println(randomHeading());
        ...
    }
}
```
### Declaration Example

- **A two line “hit counter”**
  - Multiple client requests to the same servlet result only in multiple threads calling the service method of a single service instance
  - Instance variables of a normal servlet are shared by multiple requests so accessCount does not have to be declared static

```html
<!DOCTYPE ...>
<HTML>  <BODY>
<H1> JSP Declarations </H1>
<%! private int accessCount=0; %>
<H2> Accesses to page since server reboot: </H2>
<%= ++accessCount %>  //out.println
</BODY></HTML>
```
JSP Declarations and Predefined Variables

• **Problem**
  - The predefined variables (request, response, out, session, etc.) are _local_ to the _jspService_ method. Thus, they are not available to methods defined by JSP declarations or to methods in helper classes. What can you do about this?

• **Solution: pass them as arguments**

  ```java
  <%!
  private void someMethod(HttpSession s) {
    doSomethingWith(s);
  }
  %>

  <% someMethod(session); %>

• **Note that the println method of JspWriter throws IOException**
  - Use “throws IOException” for methods that use println
Comparing Expressions, Scriptlets and Declarations

• **Task 1**
  - Output a bulleted list of five random ints from 1 to 10.
    - Since the structure of this page is fixed and we use a separate helper class for the randomInt method, **JSP expressions** are all that is needed.

• **Task 2**
  - Generate a list of between 1 and 10 entries (selected at random), each of which is a number between 1 and 10.
    - Because the number of entries in the list is dynamic, a **JSP scriptlet** is needed.

• **Task 3**
  - Generate a random number on the first request, then show the same number to all users until the server is restarted.
    - Instance variables (fields) are the natural way to accomplish this persistence. Use **JSP declarations** for this.
package coreservlets; // Always use packages!!

/** Simple utility to generate random integers. */

public class RanUtilities {

    /** A random int from 1 to range (inclusive). */
    public static int randomInt(int range) {
        return(1 + ((int)(Math.random() * range)));
    }

    public static void main(String[] args) {
        int range = 10;
        try {
            range = Integer.parseInt(args[0]);
        } catch(Exception e) { // Array index or number format
            // Do nothing: range already has default value.
        }
        for(int i=0; i<100; i++) {
            System.out.println(randomInt(range));
        }
    }
}
<DOCTYPE ...>
<HTML>
<HEAD>
<TITLE>Five random numbers between 1 and 10</TITLE>
<LINK REL=STYLESHEET HREF="JSP-Styles.css" TYPE="text/css">
</HEAD>
<BODY>
<H1>Random Numbers</H1>
<UL>
  <LI><%= coreservlets.RanUtilities.randomInt(10) %>
  <LI><%= coreservlets.RanUtilities.randomInt(10) %>
  <LI><%= coreservlets.RanUtilities.randomInt(10) %>
  <LI><%= coreservlets.RanUtilities.randomInt(10) %>
  <LI><%= coreservlets.RanUtilities.randomInt(10) %>
</UL>
</BODY></HTML>
Task 1: JSP Expressions

Random Numbers

- 1
- 3
- 10
- 9
- 6
<!DOCTYPE ...>
<HTML>
<HEAD>
<TITLE> Randomly generates a number between 1 and 10 and generate that amount of random numbers </TITLE>
<LINK REL=STYLESHEET HREF="JSP-Styles.css" TYPE="text/css">
</HEAD>
<BODY>
<H1>Random List (Version 1)</H1>
<UL>
<% int numEntries =
    coreservlets.RanUtilities.randomInt(10);
for(int i=0; i<numEntries; i++) {
    out.println("<LI>" +
        coreservlets.RanUtilities.randomInt(10));
}
%>
</UL>
</BODY></HTML>
Task 2: JSP Scriptlets
(Result: Version 1)

Random List (Version 1)

- 8
- 5
- 6
- 9
- 5
- 3
- 7
- 9
- 1
<!DOCTYPE ...>
<HTML>
<HEAD>
<TITLE> Randomly generates a number between 1 and 10 and generate that amount of random numbers </TITLE>
<LINK REL=STYLESHEET HREF="JSP-Styles.css" TYPE="text/css">
</HEAD>
<BODY>
<H1>Random List (Version 2)</H1>
<UL>
<% int numEntries =
coreervlets.RanUtilities.randomInt(10);
for(int i=0; i<numEntries; i++) {

<LI><%= coreervlets.RanUtilities.randomInt(10) %>
<% } %>

// becomes an out.println
</UL>
</BODY></HTML>
Task 2: JSP Scriptlets
(Result: Version 2)

Random List (Version 2)

- 3
- 2
- 4
- 4
- 8
<!DOCTYPE ...>
<html>
<head>
<title>Generate a random number on the first request and show the same number to all users until the server is restarted</title>
<link rel=stylesheet href="JSP-Styles.css" type="text/css"/>
</head>
<body>
<%!
private int randomNum = coreservlets.RanUtilities.randomInt(10);
%
<h1>Semi-Random Number:<br>%= randomNum %></h1>
</body>
</html>
Task 3: JSP Declarations (Result)

- Instance variables are initialized only when the object is built; servlets are built once and remain in memory between requests until the server is restarted.
- JSP expressions and scriplets deal with code inside the _jspService method.

Semi-Random Number: 2
JSP Declarations: Result

JSP Declarations

Accesses to page since server reboot: 10
JSP Declarations: the jspInit and jspDestroy Methods

- JSP pages, like regular servlets, sometimes want to use init and destroy
- Problem: the servlet that gets built from the JSP page might already use init and destroy
  - Overriding them would cause problems.
  - Thus, it is illegal to use JSP declarations to declare init or destroy.
- Solution: use `jspInit` and `jspDestroy`.
  - The auto-generated servlet is guaranteed to call these methods from init and destroy, but the standard versions of `jspInit` and `jspDestroy` are empty (placeholders for you to override).